Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 31, 2006

J. Whitehead
U.C. Santa Cruz
February 27, 2006

Design Considerations for State Identifiers in HTTP and WebDAV
draft-whitehead-http-etag-00

Status of this Memo

Copyright Notice

Abstract

This document discusses design considerations for state identifiers in the Hypertext Transfer Protocol (HTTP) and related protocols such as WebDAV.

Editorial Note

Discussion of this draft and comments to the editors should be sent to the ietf-http-wg@w3.org [1] mailing list, which is archived at

<http://lists.w3.org/Archives/Public/ietf-http-wg/>.

Table of Contents

1. Introduction

In distributed systems such as the World Wide Web, it is useful to assign unique identifiers to individual states of network resources. The basic idea is that each time the state of a network resource changes, its associated state identifier changes too. State identifiers have the quality that they uniquely identify a particular state of a network resource; only one identifier will ever be associated with a given network resource state. These state identifiers can be used to support caching and remote authoring of network resources.

In caching, each cache locally stores a state identifier along with its cached copy of a network resource, and uses the state identifier to query whether the cached copy is up-to-date. If the cache's local state identifier is different from the current state identifier of the original network resource, it indicates the cached copy is stale, and needs to be refreshed. If the cached and original state identifiers are the same, it indicates the local copy is up-to-date.

In remote authoring, a remote authoring tool wishes to update the state of a network resource. A typical authoring session involves retrieving the current state of the network resource, some editing, then writing the new state of the network resource back to its original location. There are two concerns during remote authoring. One is that another author might try to modify the same network resource at the same time, leading to the lost update problem. State identifiers are used to detect this problem. The authoring application stores a local copy of the network resource's state identifier at the beginning of the authoring session. When the application goes to write the new resource state, it compares the local state identifier with the current state identifier of the network resource. If they are the same, the network resource has not been modified, and the write can proceed without danger of overwrite. The second concern is the remote authoring client wants to know that the network resource has been stored in the same form it was submitted. There are many document formats, such as XML, that can remain semantically equivalent in the face of multiple kinds of changes to the actual octets stored. The authoring application would like to know if it can continue to use its local copy of the network resource, or if it instead needs to reload its local copy from the original.

1.1. Entity Identifiers in HTTP

Version 1.1 of the Hypertext Transfer Protocol (HTTP) [RFC2616] supports two identifiers, the Entity Tag (Etag) and the Content-MD5 hash (MD5-hash). Etags are used in HTTP 1.1 for caching, and the Web

Distributed Authoring and Versioning (WebDAV) authoring protocol uses Etags in conjunction with locks to avoid the lost update problem and keep local client state in synch with the authoring server. The MD5-hash is used to perform end-to-end message integrity checks. A key difference between Etags and MD5-hashes is the Etag is not required to be computable from the contents of the on-the-wire representation of a resource, while an MD5-hash is. A consquence is that Etags are less computationally inexpensive to produce than MD5-hashes.

Within HTTP, there are no provisions for directly interacting with the state of a network resource. Instead, clients can retrieve representations of a network resource using the GET method, and can write representations using PUT. The representation retrieved using GET can be the result of an arbitrary computational process, or can be the result of applying a wide range of transformations to a persistently stored resource. Similarly, a server may apply a range of transformations to a representation submitted with PUT before creating a persistently stored resource. A resource representation on the wire is known as an entity, and both Etags and MD5-hashes are unique identifiers for this entity.

In the most general case, Etags and MD5-hashes are not state identifiers, since they uniquely identify only for the on-the-wire representation of a resource, and do not necessarily identify the actual resource state. Etags and MD5-hashes are entity identifiers. For caching, this is usually acceptable, since the cache only wants to ensure that the client-visible representation of the resource is maintained up-to-date.

1.2. Problems with Entity Identifiers as Substitute State Identifiers

For authoring, the distinction between entity identifiers and state identifiers is problematic. Since authoring applications modify the state of the network resource, the information provided by entity identifiers does not provide sufficient feedback on the progress of authoring applications. Changes to the state of a resource must be inferred from changes in entity identifiers.

Adding to this core difficulty are many accidental ones. HTTP does not clearly specify the behavior of Etags and PUT. There is no requirement that a successful PUT response return an Etag or Content-MD5 header (it is a MAY level requirement for just one of 3 possible response codes). While many servers do return the Etag header, this is not universal. If no Etag is received in the PUT response, clients must perform an additional request to retrieve it. Unfortunately, there is no mechanism clients can use to determine if the retrieved Etag represents the one assigned to the PUT entity they submitted, or the Etag of an entity submitted subsequently by another

client. Since an arbitrary transformation could have taken place on the PUT entity before it was persistently stored, even an MD5-hash would be unreliable. This is moot, since there is no mechanism that clients can use to force a server to return an MD5-hash.

The HTTP specification is also unclear on what Etag should be returned in the response to a PUT. The current specification states that for a 201 Created response, "indicating the current value of the entity tag for the requested variant just created" (Section 10.2.2), while there is no statement concerning use of Etag with 200 or 204 responses, the other possible responses to a successful PUT. Given the specification ambiguity, it is conceivable that servers might return the Etag for the submitted entity, rather than the Etag for the current GET response for the resource.

Several HTTP servers use filesystem last modified timestamps as their mechanism for computing Etags. This has the advantage of fast recall; a simple system call retrieves a resource Etag, and does not require any computation on the state of the resource itself, or retrieval of a precomputed Etag from a database. However, since servers can process multiple write operations within the time span of the minimum granularity of the operating system clock, such servers return a provisional Etag immediately, and then upgrade this to a permanent Etag later. This requires clients to perform an additional network request to retrieve the final version of the Etag. Unfortunately, there is no mechanism clients can use to distinguish between the Etag having been changed due to promotion to a permanent Etag, or the Etag having been changed due to another authoring client modifying the resource. As before, MD5-hashes are unreliable.

This combination of the essential different between state identifiers and entity identifiers, and the several accidental difficulties in specifying and implementing entity identifiers have combined to create substantial difficulty for authoring clients using the WebDAV protocol. These difficulties make it impossible, in the general case, for authoring clients to have any confidence that they have successfully written an updated resource to a remote server. Since this the core operation supported by remote authoring clients, this problem has broad ramifications for the adoption and use of HTTP-based remote authoring.

In the remainder of this document we describe the requirements for clients and servers for state and entity identifiers. When then document several current behaviors by HTTP servers that contribute to the difficulty of using the current entity identifiers. Following, we note several specification ambiguities that contribute to the problem. We then outline the characteristics of a broad solution to the problem. Our goal is for this document to be used as a statement

of goals for a subsequent protocol specification that substantially addresses the concerns raised herein.

2. Requirements for State Identifiers and Entity Identifiers

Three scenarios that drive requirements for state identifiers and entity identifiers are caching, end-to-end message integrity checks, and authoring. Additionally, implementation concerns also provide requirements.

2.1. Caching Requirements

The following two requirements drive the existence of weak and strong Etags. While the complete set of requirements for HTTP caches is quite broad, the requirements below are the ones specifically related to entity identifiers and state identifiers.

A client must be able to determine if its cached copy of the GET response for a resource is octet-for-octet the same as the current GET response, without having to re-retrieve the current GET response.

A client must be able to determine if its cached copy of the GET response for a resource is semantically equivalent to the current GET response, without having to re-retrieve the current GET response. Two responses might be considered semantically equivalent even if not octet-for-octet equivalent if, for example, they had minor differences in HTML encoding, or some automatically updated value like a hit counter was considered semantically irrelevant.

2.2. End-to-End Integrity Check Requirements

The following requirement drives the existence of the MD5-hash.

A client or server must be able to determine if an HTTP message has been transmitted through zero or more intermediaries without modification to the entity body.

2.3. Authoring Requirements

An authoring client must be able to determine if the state of a resource at the beginning of an editing session remains unchanged when the client wishes to update the state of the resource.

An authoring client must be able to determine if the server has made changes to an entity submitted during PUT that would require the client to reload the resource to have the correct current state. This determination must be reliable, in the sense that the client

must be able to receive an unambiguous answer to the query, "has the server modified the submitted entity prior to its persistent storage?"

An authoring server must be able to modify the entity submitted using PUT before persistently storing it. Servers frequently modify submitted data. Examples based on current applications include modifying XML to change XML namespace usage, change linear whitespace, and sometimes modify the character encoding. Versioning servers also may perform keyword expansion in the body of submitted source code, e.g., to inject the author, version identifier, date, etc. Calendar servers may annotate calendar event resources with server-specific properties.

An authoring client must be able to direct the server to reject a request to persistently store the resource if it cannot guarantee octet-for-octet storage of the submitted entity. This requirement is more speculative than the others, since it does not describe a strongly expressed existing client need. Still, there are many media types that cannot withstand any server tampering, such as the native formats of many kinds of application software that store their documents in a proprietary binary format. Any server tampering with these document types would corrupt the document.

WebDAV resources have two types of state, the resource body, a representation of which is returned by GET, and resource properties, a representation of which is returned by PROPFIND. An authoring client must be able to determine if changes have occurred to entries in both kinds of state. State identifiers must not mix state types. That is a state identifier for the resource body should be independent of state identifiers for properties. An open question is the necessary granularity of state identifiers for properties.

2.4. Implementation Driven Requirements

Since GET is a very common operation, it must be possible for servers to efficiently compute any entity or state token returned by GET. Alternately, it must be possible for servers to not return expensive-to-compute identifiers unless specifically requested by the client.

The response to any write operation must return state identifiers and entity identifiers associated with the permanent persisted state of the resource following the operation. This is the only reliable mechanism for communicating this information to the client.

3. Current Implementation Behaviors and their Implications

_To do:_

1. Document the Apache server behavior of returning a weak etag with the PUT response then promoting this to a strong etag. Note that this makes it impossible for clients to reliably determine the permanent etag associated with the resource.

2. Document a server that performs significant content modification upon PUT (a CalDAV server?)

3. Others?

4. Ambiguities in the HTTP and WebDAV Specifications

4.1. Confusion over the meaning of the Etag returned in a PUT response

It is currently unclear as to which entity is identified by an Etag returned in the response to a PUT.

The HTTP specification [RFC2616] states (Section 10.2.2):

> "A 201 response MAY contain an ETag response header field indicating the current value of the entity tag for the requested variant just created, see section 14.19."

Hence, for situations where a new resource is created, the meaning of Etag is clear.

The HTTP specification also states (Section 14.19):

> "The ETag response-header field provides the current value of the entity tag for the requested variant."

Since the other success responses for a PUT request (200 and 204) provide no specification for the meaning of Etag, a strict reading of the specification is ambiguous, since there is no "requested variant" here. Presumably the 201 Etag semantics are intended for 200 and 204 responses to PUT, though this is not explicitly stated in the HTTP specification.

Another possible interpretation is that the server should return the Etag of the entity just submitted by the client in the PUT request. In the section below, we describe an ambiguity where the Etag returned may be expected to vary depending on the amount of processing the server performs on the submitted entity.

4.2. Confusion over Semantics of Strong Etags

The HTTP specification states (Section 3.11):

> A "strong entity tag" MAY be shared by two entities of a resource only if they are equivalent by octet equality.

When a client performs a PUT, there are two entities in play:

A. the entity submitted by the client in the initial PUT request

B. the entity returned by the server in subsequent GET requests

A question that arises is whether a server can return a strong Etag if it modifies the submitted entity, A, before persistently storing it. Supporting a yes viewpoint, we note that the submitted entity, A, isn't an entity of the resource until the PUT operation succeeds, because only the success of the operation associates it with the resource. As a result, it is not reasonable to discuss equivalence of A and B as two entities of the same resource, since A is not yet associated with the resource.

Supporting a no viewpoint, we note that the intent of the Etag is to act as an entity identifier. If we perform two GET operations in a row on a resource, and we receive the same strong Etag in each response, we expect two response entity bodies to be octet-for-octet the same. Hence, if we submit an entity, A, and receive a strong Etag in return along with entity B, there is an assumption that the submitted entity has not been modified, and A is octet-for-octet the same as B. We note that there is no language in the HTTP specification to support this viewpoint.

5. Dimensions of a Solution

TBD.

There are currently three suggestions.

5.1. Julian's suggestion

Alternative 1: Make strong server requirement; i.e., mandate to only return ETag if content was written octet-by-octet. Drawback: this in not required in HTTP, thus potentially implemented differently in existing servers. No way for a client to tell the difference. Also, returning strong ETags although content rewriting happens may have its use cases; it only becomes a problem if the client tries to use the ETag as cache validator in a byte-range request (which the server

could reject).

Alternative 2: Add a new Response Header through which servers can indicate whether they need to refetch content or not. Note that header would not have a default, so clients can simply detect whether they speak to "new" server. This would also be applicable to other write methods, such as PROPPPATCH: for instance, would a PROPPATCH affect the representation of the resource (i.e., metadata is stored in body such as in JPEG, MP3, Office docs...), the server could return a new ETag and indicate that the entitiy changed.

[[anchor15: jr -- For reasons of compatibility with existing implementations, the second alternative seems to be superior to me]]

5.2. Lisa's suggestion

Thus, we RECOMMEND servers supporting ETag and PUT return the ETag header in the PUT response, and we RECOMMEND clients receiving the ETag in a PUT response use their local copy of the resource rather than query the server for a redundant copy.

When a client does not receive an ETag header at all in a PUT response, the client MUST NOT consider its local copy of the resource to be up-to-date with the server's copy.

The Get-ETag response-header field provides the value of the entity tag for the entity of the resource that would be provided on a subsequent GET request.

```
Get-ETag = "Get-ETag" ":" entity-tag
```

The Get-ETag header is appropriate for use when the server can only guarantee that it can return the entity with that tag in response to a GET, not an entity that is byte-for-byte equivalent to the entity the client provided.

5.3. Jim's suggestion

Introduce the notion of a resource body state identifier that uniquely identifies the persistently recorded state of a resource. Introduce the notion of a resource property identifier that identifies the aggregate persistently recorded state of all dead properties. Then, introduce six new headers, and two new properties:

Resource-State: a response header that indicates the current state identifier for the resource after successful performance of the requested operation. In the case of an error, it indicates the state of the resource prior to the failed operation. This header is only

included in a response if the client specifically requests it using the Request-Resource-State header.

Property-State: a response header that indicates the current property state identifier for the resource after successful performance of the requested operation. In the case of an error, it indicates the state of the resource prior to the failed operation. This header is only included in a response if the client specifically requests it using the Request-Property-State header.

Request-Resource-State: a request header used to request a Resource State header in the response. May be used with any method.

Request-Property-State: a request header used to request a Property State header in the response. May be used with any method.

Content-Handling: a response header the MUST be returned by a successful PUT. Broadly indicates the kind of handling performed by the server when storing the submitted entity. Acceptable values are "none" (entity was stored octet-for-octet), "XML" (processing that modified the entity but did not change the semantics according to XML rules, only applicable to XML content types), "some" (the server performed some modification of the entity), "encoding" (the server changed the entity's content encoding only). Allow for extensions to this set of values.

Acceptable-Content-Handling: a request header usable in PUT requests only. A list of tokens from the set defined with Content-Handling. If the response Content-Handling header value is not one of the tokens listed in this header, the request MUST fail.

Add a new value to the DAV header, "fixed-PUT" to indicate support for these semantics.

Two new properties, one to represent the resource state identifier, and the other to represent the aggregate property identifier.

6. References

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[1] <mailto:ietf-http-wg@w3.org>

Author's Address

Jim Whitehead
UC Santa Cruz, Dept. of Computer Science
1156 High Street
Santa Cruz, CA 95064

Email: ejw@cse.ucsc.edu

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).